Máster en Ingeniería Informática

Equilibrado de juegos multijugador mediante generación procedimental de contenido

Convocatoria de febrero: Notable 7

Miguel Andrés Herrero

Departamento de Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad Complutense de Madrid

Directores:

Prof. Dr. Marco Antonio Gómez Martín,

Prof. Dr. Pedro Pablo Gómez Martín

ÍNDICE

RESUMEN 7

ABSTRACT 8

CAPÍTULO 1.- INTRODUCCIÓN 9

1.1.- Objetivos y plan de trabajo 12

CHAPTER 1.- INTRODUCTION 13

1.1.- Objectives and work plan 16

CAPÍTULO 2.- ESTADO DEL ARTE 17

2.1.- Fases del desarrollo de videojuegos 18

2.2.- Roles 19

2.3.- Equilibrado 20

2.4.- Generación procedimental de contenido 22

2.5.- Equilibrado de juegos por generación procedimental 24

2.6.- Herramientas utilizadas para el desarrollo del proyecto 24

2.6.1.- Unity 3D 24

2.6.2.- Visual Studio 26

2.6.3.- BitBucket 26

CAPÍTULO 3.- IMPLEMENTACIÓN DE UN SISTEMA DE NIVELADO 27

3.1.- Qué es una biblioteca 29

3.2.- Estructura de la biblioteca 29

3.3.- Implementación de la biblioteca en Unity 3D 33

CAPÍTULO 4.- JUEGOS DE PRUEBA 37

4.1.- Runner procedimental 37

4.2.- Juego arcade procedimental 42

4.3.- Evaluación empírica 44

CAPÍTULO 5.- EVALUACIÓN 47

5.1.- Realización de la evaluación 47

5.2.- Interpretación de resultados 51

5.3.- Trabajo futuro 51

CAPÍTULO 6.- CONCLUSIONES 53

CHAPTER 6.- CONCLUSIONS 55

REFERENCIAS 57

El/la abajo firmante, matriculado/a en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Equilibrado de juegos multijugador mediante generación procedimental de contenido", realizado durante el curso académico 2017-2018 bajo la dirección de Marco Antonio Gómez Martín y Pedro Pablo Gómez Martín en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en internet y garantizar su preservación y acceso a largo plazo.

Marco Antonio Gómez Martín Pedro Pablo Gómez Martín

Miguel Andrés Herrero

Agradecimientos

No hay nada que esté enteramente en nuestro
poder más que nuestros pensamientos

René Descartes

Desde que empecé el grado en la Facultad de Informática hasta ahora,que acabo el Máster en Informática, habiendo pasado también por el de videojuegos, han pasado muchos años en los que he podido conocer a personas increíbles, que han aportado mucho para que hoy esté aquí.

Lo primero, quiero agradecer a mis padres y mi hermana el haber estado ahí cuando los necesitaba. A mi padre por sus llamadas cada día, preguntándome como estaba y si necesitaba algo. Siempre que ha estado en su mano me lo ha dado. A mi madre, con la que convivir me ha hecho sentir que tenía a alguien a diario con quien compartir todo. Su ánimo día a día ha conseguido que siga adelante. Se que movería cielo y tierra si hace falta para ayudarme. A mi hermana, por animarme siempre en algunos momentos malos estos años. Quiero hacer también un agradecimiento especial a mis abuelos, que siempre me han escuchado, aconsejado, y tratado como a un hijo suyo. Y eso nunca podré compensárselo.

Después de mi familia, a mis directores, Marco y Pedro. Mi primer contacto con vosotros fue en el Máster de Videojuegos, donde enseñábais con ilusión y nos animábais a perseguir nuestros sueños. Sin vuestro apoyo y lo que me habéis empujado estos meses, este trabajo no estaría ahora aquí. Gracias por vuestra empatía, y saber utilizar el sentido del humor, desdramatizando momentos del proyecto que me agobiaba. No he podido tener mejores directores, y no creo que haya mejores profesores que vosotros en la carrera. Vuestro modo de guiar y enseñarme me ha encantado.

Debo hacer también una mención especial a la familia que tengo en la Facultad, que es ASCII. Desde que entré ha sido parte de mi día a día de un modo muy intenso. Ellos me han hecho reír y desconectar en cada momento del día que lo necesitaba. Han logrado distraerme cuando era necesario, y desesperarme cuando también lo era. Pero ante todo han conseguido eso, ser una familia.

Por último, y no menos importante, a unos pocos amigos en particular. Pont, con su lógica absurda que siempre me saca una sonrisa. Ray, con sus charlas dia a dia

sobre algo interesante. Vik y Juan, que han hecho estos años del Máster entretenidos y amenos.

David, Pascal, por todo lo que he aprendido de ti, y aún me queda. Por estar siempre ahí. A Keno, por sacarme de casa cuando estaba a punto de desesperarme o simplemente reírnos, y por toda la experiencia que me transmites y aconsejas.

A Jorge (o Hippie como prefieras), que después de todos estos años juntos, de estar ahí siempre para todo, y de cómo nos entendemos, eres como un hermano para mí.
A Blanca, por hacer mi último año y medio mucho más fácil, por su preocupación y por confiar en mí cuando ni yo mismo era capaz. Y por empujarme hasta el final para acabar este trabajo. Una parte de el es tuyo.

Esto es por todos vosotros.

RESUMEN

En los últimos años, la industria de los videojuegos se está decantando por potenciar un campo poco explotado en el área y que puede ofrecer una gran versatilidad para el desarrollo: la generación procedimental.

Este método ha conseguido ofrecer nuevas alternativas al contenido de un juego y rebajar la carga de diseño que los mismos necesitan. Sin embargo, aun quedan varias aplicaciones sin explotar. Una de ellas, en la que nos centraremos en este trabajo, es la generación procedimental aplicada al equilibrado de juegos en tiempo real. Hasta donde se ha documentado, no hay ninguna referencia de que nadie haya intentado aplicarla de este modo.

Es esto mismo lo que ha motivado el intentar realizar una aplicación distinta de la generación procedimental para que pueda servir como una toma de contacto en futuros trabajos sobre la viabilidad del equilibrado de videojuegos en tiempo real.

En la siguiente memoria se muestra los resultados obtenidos y a la vez expondremos una biblioteca para poder simplificar la implementación de este equilibrado en juegos multijugador.

Palabras clave: generación procedimental, equilibrado de videojuegos, videojuego multijugador.

ABSTRACT

During the last few years, the videogame industry is opting to promote a field that is little exploited in the area and that can offer great versatility for development: procedural generation.

This method has managed to offer new alternatives to the content of a game and to reduce the design load that they need. However, there are still several untapped applications. One of them, in which we will focus on this work, is the procedural generation applied to the balancing of games in real time. As far as it has been documented, there is no reference to anyone trying to apply it in this way.

This is what it has motivated the attempt to make a different application of the procedural generation so that it can serve as a contact point on the feasibility of balancing video games in real time in future works.

The following report shows the results obtained and at the same time we will expose a library in order to simplify the implementation of this balancing in multiplayer games.

Keywords: procedural generation, balancing of videogames, multiplayer games.

CAPÍTULO 1.- INTRODUCCIÓN

Desde la aparición de los primeros juegos para ordenador en 1960, la industria del videojuego se encuentra en un proceso de constante expansión[1]. Gracias a la evolución que ha experimentado este sector a lo largo de los años, de la mano de nuevas tecnologías, podemos diferenciar distintas etapas.

Los primeros años de la industria vienen marcados por juegos puntuales [1]. En las décadas de los 50 y los 60 los videojuegos eran hardware, hasta que comienzan a aparecer los primeros juegos desarrollados con lenguaje ensamblador a partir de 1970. Los videojuegos eran poco más que caracteres alfanuméricos y que no tenían gráficos

Es en este momento, cuando surgen las máquinas recreativas. Éstas se agrupaban en los conocidos salones recreativos en los cuales se podía jugar una partida a varios juegos a cambio de una moneda.

El nacimiento de la industria se considera en este punto debido a que, ante el éxito de las máquinas recreativas de las salas, Atari, una de las empresas más potentes del momento, toma la decisión de comenzar a fabricar sus propias máquinas, surgiendo así las primeras videoconsolas.

Éstas estaban específicamente diseñadas para reproducir los videojuegos de la compañía. Esto coincide en el tiempo con la irrupción del ordenador doméstico en las viviendas, para los cuales también se comienzan a desarrollar algunos juegos. A esta generación se la conoce como la generación de los 8 bits.

Esto supuso un cambio en el enfoque del mercado, pues ahora los desarrolladores podían hacer juegos y distribuirlos a tiendas para que los propietarios de las consolas los comprasen.

Toda esta evolución poco a poco fue haciéndose más notable debido a que los lenguajes de programación también evolucionaban. Se comenzaron a programar juegos en C y se empezaron a incluir los primeros recursos gráficos (sprites) y sonoros.

---

1 http://www.expansion.com/economia-digital/companias/2017/08/26/5996fc9722601d6d3b8b4598.html

A la generación de los 8 bits le sucedió la de los 16 bits debido a la evolución constante del hardware. En muchos hogares además ya se disponía de un PC, y por ende, se desarrollaban juegos para el mismo.

Esta evolución hizo que en pocos años llegase en la llamada generación de los 32 bits la guerra de las consolas donde grandes empresas como Nintendo o Sony se fueron haciendo con el mercado.

Con esta evolución, el videojuego también cambió. Los primeros videojuegos contaban con recursos gráficos escasos. Tenían bajas resoluciones y poca capacidad de procesamiento gráfico, por lo que dejaban una gran parte a la imaginación del usuario.

Sin embargo, al aumentar la potencia del hardware y evolucionar los lenguajes de programación, los juegos fueron añadiendo cada vez más gráficos llamativos. La programación introdujo cambios nuevos, como la programación orientada a objetos, que permitía una mayor abstracción para este campo (C++ a finales de los 90).

De este modo, comienzan a surgir juegos cada vez más completos y de distintos géneros: deportivos, arcade, RPG, aventura… La industria comenzaba a crecer rápido de la mano de la tecnología. Además del hardware y los lenguajes de programación, también evolucionó el estilo de programación.

El modelo de desarrollo de videojuegos también cambió. Al introducirse recursos gráficos y de audio en los mismos, se abrió un nuevo abanico de posibilidades. A esto sumamos que comienza a separarse el diseño de la programación de juego y aparece la estructuración de los propios juegos mediante niveles, desarrollados para el motor.

Los niveles permitían dividir el juego en partes más pequeñas, en las que incluir recursos específicos e, incluso, funcionalidad.

Sin embargo, hay un momento del desarrollo de la industria, donde ya había muchos títulos publicados, en el que surge la idea de hacer juegos que ofreciesen contenido jugable distinto cada vez que fuesen jugados.

Es así como parte de la industria decide investigar sobre cómo generar ese contenido en tiempo de juego. Aun así, los primeros pasos se dieron a los pocos años de la aparición de los videojuegos con Beneath Apple Manor en 1978[2].

[2] https://en.wikipedia.org/wiki/Beneath_Apple_Manor

A partir de este primer precursor, se comienza a experimentar con lo que hoy conocemos como generación procedimental. Este concepto consigue ganar adeptos y se empieza a aplicar en distintos campos del videojuego, mapas, lógica de toma de decisiones puntuales, etc. En resumidas cuentas, se busca su explotación para poder generar contenido nuevo para los videojuegos.

Surgen géneros como el *Roguelike* (videojuegos que se caracterizan por su dificultad y riqueza en contenido, similares al videojuego del cual proviene el nombre, Rogue[3]), que se jacta del uso de la generación procedimental de contenido para que sus juegos sean siempre distintos en varias cosas y poder decir así que pueden ser rejugados varias veces sin aburrirse.

Esto tiene a su vez un gran tirón en los consumidores de contenido audiovisual de la industria, hasta tal punto que hoy se demandan títulos con estas características y posibilidades. Además, en los últimos años, se está trabajando cada vez más en títulos que trabajen con este tipo de algoritmos para generar contenidos.

Muchas empresas de videojuegos están invirtiendo grandes esfuerzos en conseguir perfeccionar y pulir esta técnica. Además, varios portales de noticias relacionadas con esta industria afirman desde hace años que el futuro de los videojuegos pasa por la generación procedimental de contenido[4].

Otro motivo fundamental para su auge es que desde los primeros años de la industria hasta casi los 90 no se ha podido explotar este campo salvo en esos pocos intentos y para cosas muy concretas.

Sin embargo la generación procedimental de contenido suponía una gran baza a la hora de desarrollar contenido debido a que el código usado para desarrollarlo ocupaba mucho menos que el contenido generado y el único lastre que podemos observar a la hora de generar contenido de este modo es el tiempo que tarda el algoritmo en generarlo.

Aun así, la generación procedimental de contenido generaba también algo de incertidumbre. A pesar de que automatizaba la creación de niveles, no se sabía si los niveles generados serían divertidos o no.

[3] https://es.wikipedia.org/wiki/Rogue
[4] http://www.fsgamer.com/generacion-procedimental-el-futuro-de-los-videojuegos-20150130.html
https://www.xataka.com/videojuegos/procedurally-generated-content-la-revolucion-de-los-videojuegos-es-ahora-aunque-llevamos-40-anos-creandola

También había que tener en cuenta cosas como el tipo de jugadores al que iría dirigido. O por ejemplo, si estos niveles podrían resultar excesivamente fáciles o difíciles.

1.1.- Objetivos y plan de trabajo

Este proyecto se enfoca en la aplicación de la generación procedimental de contenido para un fin poco explorado hasta ahora: aplicarlo a juegos multijugador para conseguir un equilibrado entre ambos jugadores y balancear así el juego.

Y para simplificar esto, intentaremos implementar una biblioteca que pueda servir de utilidad a los desarrolladores que quieran implementar juegos de este estilo.

El objetivo que perseguimos con esto es ver si resulta útil y viable este tipo de técnicas para ver si realmente se puede conseguir un equilibrado efectivo en un videojuego sobre el mismo y durante su ejecución.
A la par, desarrollaremos una librería que permita implementar ese equilibrado de forma simple e intuitiva para reducir los costes de este desarrollo.

Para ello el plan de trabajo que seguimos durante todo el proceso fue el siguiente:

1. Búsqueda de información sobre el tema.
2. Una vez realizada la fase de documentación, decidir herramientas para su desarrollo (motor gráfico, etc...).
3. Desarrollar un juego sencillo para dos jugadores implementando un equilibrado en tiempo real sencillo.
4. Abstraer el sistema de equilibrado para que pueda funcionar de forma genérica en cualquier videojuego donde sea implementado como biblioteca reutilizable.
5. Una vez desarrollado un modelo básico de la biblioteca, desarrollar otro pequeño juego que haga uso de la misma para validar su diseño comprobando si, efectivamente, facilita la creación de videojuegos con equilibrado de dificultad automática.
6. Probar con usuarios los juegos para recoger datos de si el equilibrado es efectivo.

A lo largo de este documento, iremos viendo en profundidad como hemos llevado a cabo cada uno de estos puntos, y finalmente expondremos unas conclusiones.

CHAPTER 1.- INTRODUCTION

From the start of the first computer games in 1960, the video game industry has been in constant expansion[5]. Thanks to the development that it has experimented through the years, and thanks to new technologies too, we can classify its development in different stages.

The first few years the industry [1] were marked by specific successful games. In the both the decades of the 50s and 60s video games were made solely on hardware, it wasn't until 1970 when games started being developed in assembly language. Video games then were little more than alphanumeric characters without graphics.

It is at this moment when arcade machines make their appearance. These were placed in the known arcade salons where people could play various games in exchange of a coin.

This is considered as the birth of the video game industry because due to the success of the arcade machines, Atari, one of the most powerful business at the time, decided to start producing their own machines, that way emerging the first consoles.

Game consoles were specifically designed for playing the games from their own companies. This coincides with the surge and commercialization of computers for homes, for which they started to have video games developed for their platform too. This generation is known as the 8 bit generation.

This supposed a change of perspective in the market, for developers could make games and distribute them at shops for console owners to buy.

This evolution was made more palpable as programming languages were also evolving. Video games began being programmed in C and the first graphic (sprites) and audio resources started being included.

5 http://www.expansion.com/economia-digital/companias/2017/08/26/5996fc9722601d6d3b8b4598.html

The 8 bit generation was followed by the 16 bit one caused by the constant evolution of hardware. Also, In many homes people already had a PC, and so games were developed for them.

This evolution provoked in very few years the arrival of, with the so called 32 bit generation, the console war where the biggest businesses like Nintendo or Sony took control over the market.

With this evolution, video games changed too. The first video games that came out barely had any graphic resources. They had low resolutions and scarce graphic processing, thus they left a great deal of information to the user's imagination.

However, due to the hardware empowerment and development of programming languages, video games started adding more striking graphics. Programming introduced new changes, like object oriented programming, that allowed a bigger abstraction in this area (C++ by the end of the 90s).

In this manner, more complete and from different genres games begin emerging. Sports, arcade, RPG, adventure… the industry started growing faster by the hand of technology. Apart from hardware and programming languages, programming styles also evolved.

The video game development model also changed. Due to the introduction of graphic and audio resources in them, a new pathway of possibilities was available. To this we must add that design and programming in games were being separated and thanks to it came the appearance of level structure in games, developed with the game engine.

Levels allowed dividing games in smaller parts, which permitted including more specific resources and even functionalities.

Still, there was a moment in the development of the industry where there were already many titles published, in which appeared the idea of making games that offered different playable content each time they were played.

That is how part of the industry decided to investigate how to generate that content in in-game time. However, the first steps were taken after a few years of the appearance of video games with Beneath Apple Manor[6] in 1978.

[6] https://en.wikipedia.org/wiki/Beneath_Apple_Manor

Since this first forefather, experimentation starts on what we call today procedural generation. This concept manages to win over adepts and it starts being applied on different game areas, maps, decision making logic, etc. To sum it up, the investigation is based on the search of generating new content for video games.

Genres such as Roguelike (video games that are known for their difficulty and rich content, similar to the video game their name the own, Rogue[7]) appear, that boast in the use of procedural content generation for their games to be always different in many things and thus be able to remark that they can be replayed many times without finding them boring.

This results in an attractive product to consumers of audiovisual content in the industry, to the point in which today titles with such characteristics and possibilities are on demand. Also, in the last few years, it is being more common to work in titles that use this type of algorithm to generate content.

Many video game companies are investing a great deal of effort on getting to perfect and polish this technique. Besides, several news portals related to this topic confirm that for years the future of video games is based on procedural content generation[8].

Another fundamental reason for its peak is that from the first years of the industry to nearly the 90s it has not been able to exploit this area except for the few tries that were made and for very concrete things.

Nevertheless the procedural content generation supposed a great bet in developing content because the code used to develop it occupied much less than the generated content and the only encumbrance we can observe when generating content like this is the time the algorithm takes to generate it.

Even then, procedural content generation produced some uncertainty. Even though it automated level creation, it wasn't known if the generated levels would be fun or not.

Also, things like to which players it would be centered on had to be considered. Or for example, if these levels could result on being excessively easy or hard.

[7] https://es.wikipedia.org/wiki/Rogue

[8] http://www.fsgamer.com/generacion-procedimental-el-futuro-de-los-videojuegos-20150130.html
https://www.xataka.com/videojuegos/procedurally-generated-content-la-revolucion-de-los-videojuegos-es-ahora-aunque-llevamos-40-anos-creandola

1.1.- Objectives and work plan

This project is focused on the application of procedural content generation with a very unexplored aim to the date: apply it to multiplayer games to obtain a balance among both players and thus, adjust the game to them.

To simplify it, we'll try to implement a library that can be used by developers who want to make games in such style.

The goal we are after is to see if it is of utility and if it is viable in this type of techniques to see if an effective balance really is achievable in a video game itself and during its execution.
At the same time, we'll develop a library that allows implementing this balance in a simple and intuitive way to reduce developing costs.

For this, the plan that we followed during the whole process was:

1. Information research about the topic.
2. Once the research phase was done, decide which tools were to be used to develop it (graphic engine, etc...).
3. Develop a simple game for two players implementing a balancing tool in in-game time.
4. Abstract the balancing system for it to work in a generic way in any video game where it be implemented as a reusable library.
5. Once having developed a basic model of the library, develop another small game that uses said library to validate its design proving if, indeed, it eases the creation of video games with a difficulty automated balancing.
6. Test the games with users to gather data to see if the balancing tool is efficient.

Through this document, we'll be seeing in depth how we have managed each of this points and, finally, we'll show our conclusions.

CAPÍTULO 2.- ESTADO DEL ARTE

La industria de los videojuegos se encuentra en continuo desarrollo debido a la gran demanda de contenido audiovisual por parte de los usuarios [2]. Esto ha hecho que el proceso de desarrollo de este software se vuelva cada vez más largo, costoso y complejo.

La duración y coste de un videojuego es muy variable. A un lado del espectro se sitúan los juegos conocidos como "triple A", mientras que al otro lado se sitúa el conocido como desarrollo Indie.

Los triple A son aquellos juegos desarrollados por grandes empresas y corporaciones. Estos, tienen equipos de cientos de trabajadores y sus producciones tienen costes millonarios, llegando a millones de euros.

Dentro de los ejemplos más conocidos, podemos encontrar Star Wars The Old Republic de BioWare[9], lanzado en 2011 y con un coste de entre los 150 y los 200 millones de dólares.

Los videojuegos de desarrollo Indie (también conocido como desarrollo independiente) son producidos por pequeñas start-ups y empresas donde su número de trabajadores es muy reducido y muchas veces la misma persona asume varios roles dentro del proceso de desarrollo del videojuego. Este proceso se ve limitado entre otras cosas por los costes debido a que el presupuesto de las empresas que los desarrollan es bajo.

Aun así, no quiere decir que sea barato. Encontrar referencias de coste de desarrollo Indie es bastante complicado, pero se cifra aproximadamente en el número de empleados, multiplicado por 10000 y por el número de meses de desarrollo[10]. Este coste luego se debe ajustar mucho, como pasó con Shovel Knight.

El desarrollo independiente comenzó en los 70[11], pero debido a restricciones en los años posteriores por parte de las distribuidoras, fue desapareciendo hasta que él los últimos 10, 15 años se ha vuelto a retomar en forma de los estudios Indie. Este repunte es en parte debido a las facilidades que se pueden tener para obtener medios para desarrollar juegos con pocos recursos.

---

[9] http://www.videojuegosadiario.com/2013/06/28/los-juegos-que-mas-costaron-para-desarrollar/
[10] http://www.anaitgames.com/noticias/cuanto-cuesta-hacer-un-juego-indie
[11] https://es.wikipedia.org/wiki/Desarrollo_de_videojuegos_independiente

Además, el desarrollo de un videojuego es complejo debido a que precisa de la coordinación tanto de técnicas de ingeniería, como de arte, diseño, planificación y marketing. Dado que es un producto que maneja una gran cantidad de recursos y que aúna a muchas personas de distintas disciplinas a trabajar juntos, hace que la coordinación del equipo sea fundamental.

2.1.- Fases del desarrollo de videojuegos

Los juegos de hoy en día han alcanzado una dimensión que no se podía imaginar hace años. En los primeros años en que surge la industria el desarrollo de un juego estaba en manos de muy pocas personas.

Pero hoy en día dada la gran cantidad de medios que tenemos a nuestro alcance y a la capacidad de formación digital que tenemos a nuestra disposición, hace que un equipo recién formado tenga la posibilidad de llevar a cabo un desarrollo.

Sin embargo, y dado a que el proceso de desarrollo de un videojuego es una tarea tan compleja, hay discrepancias entre las distintas fases y cómo estructurarlas.

En este trabajo daremos por buenas las fases descritas en [3] y por tanto consideramos que las fases por las que pasa la producción de un videojuegos son:

- Preproducción: se hace una primera aproximación con un prototipo jugable para comprobar la viabilidad del producto. Dentro de esta podemos ubicar las siguientes subfases:
    - Concepto: se comienzan a definir los aspectos básicos, como el género, gameplay, personajes base...
    - Diseño: se detallan más en profundidad elementos como la ambientación, mecánicas, sonido etc.
    - Planificación: se identifican las tareas fundamentales en el desarrollo y se estima su duración.
- Producción: desarrollo del juego en distintas fases, por parte de un equipo multidisciplinar que aglutina varias disciplinas: arte, sonido, programación, diseño…
- Testing: la realización de pruebas del producto. Las primeras pruebas se realizan con versiones conocidas como alpha, y se llevan a cabo a nivel interno en la propia empresa.
- Postproducción: en ella se ubica el mantenimiento del producto y el marketing.

2.2.- Roles

Si debemos destacar dos roles fundamentales en el desarrollo de videojuegos, que nos interesan en especial, son el del programador y el del diseñador (Figura 1).

Además de estos dos roles, es importante destacar también la labor de los artistas y músicos. Los primeros, se encargan de la parte gráfica del videojuego, estética, y su desarrollo. Los segundos, se encargan tanto de realizar los efectos de sonidos dentro del juego, como de producir la banda sonora del mismo.

El diseño se encarga de definir las reglas globales del juego. Los diseñadores deben especificar las mecánicas que debe tener y que posteriormente serán desarrolladas.

Asimismo, deberán idear los distintos niveles o fases que tenga el juego dentro de él. También tendrán que crear un sistema teórico de balanceo del juego en distintos aspectos, etc.

La labor del diseñador es una tarea que precisa ciertas bases de lógica y de una coherencia entre la arquitectura del juego que es complicado conseguir. Debe buscar un equilibrio entre la diversión, el entretenimiento y la dificultad.

Mientras que en un desarrollo independiente, esta labor es llevada a cabo por una o dos personas, en los videojuegos triple AAA son decenas de personas los encargados de buscar una armonía entre las tres características básicas de un juego citadas anteriormente.

La programación es otra parte importante, la cual tiene una profunda carga de ingeniería, métodos algorítmicos, y codificación necesarios para traducir las reglas ideadas por los diseñadores al lenguaje entendido por las máquinas. Es una tarea densa que se ve condicionada por la especificación dada por los diseñadores.

Los programadores tienen un perfil de ingeniero informático, el cual se ha especializado en áreas de informática gráfica, desarrollo y uso de motores gráficos, y con un fuerte conocimiento de programación, inteligencia artificial y algoritmos.

Al verse la tarea del programador condicionada por el trabajo del diseñador, la comunicación entre estos dos roles debe ser fluida para garantizar una buena productividad y un trabajo correcto.

Esto es importante debido a que se puede dar la situación en que el programador interpreta de un modo incorrecto el diseño dado e implementa una lógica que no es la esperada. Cuando esto se produce, es importante que se corrija rápidamente, porque si no es así, todo el trabajo hasta ese punto puede no ser reutilizado.

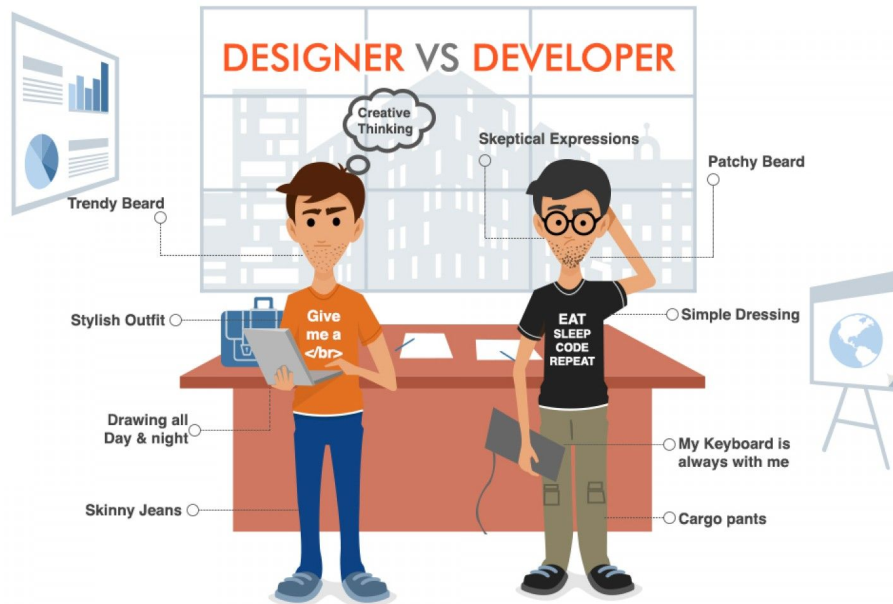

Figura 1: Imágen que ilustra los dos perfiles y sus diferencias de un modo sarcástico

2.3.- Equilibrado

El equilibrado en un juego es el proceso por el cual se busca una homeostasis entre el jugador y el producto para que no resulte demasiado difícil, ni demasiado fácil.

Una vez el juego comienza a tener prototipos jugables, conocidos como alphas, el juego tiene que ser equilibrado. Esto se consigue mediante diseño, pruebas de ensayo error, y traduciendo después los resultados al código que maneja el sistema.

Esta, es una de las tareas más duras, y que lleva más tiempo en un desarrollo. Es necesario que se realice cuanto lo antes posible en el proceso de desarrollo, y esto se ve condicionado a tener prototipos jugables pronto, que permitan al equipo usarlos y comenzar a ver posibles fallos, etc.

Como hemos visto antes, el balanceo se puede hacer mediante diseño. Por ejemplo, en un juego que enfrente enemigos, estilo Pokemon, el diseño se encarga de establecer un equilibrio entre los distintos tipos para que ninguno sea más poderoso que otro.

De este modo, diseña unos monstruos que clasifica en tipos, los cuales son fuertes frente a otros monstruos en concreto. Sin embargo, para conservar el equilibrio y balancearlos, también hace que sean débiles hacia otros.

Figura 2: Captura de pantalla del juego Crash Team Racing para PlayStation

Otro ejemplo claro de balanceo por diseño es en los juegos de carreras, como Mario Kart o Crash Team Racing (Figura 2). En estos juegos, los jugadores compiten pilotando distintos tipos de coches definidos mediante una serie de atributos que influyen a lo largo de la carrera (velocidad, aceleración, giro…). De este modo, para que ninguno de los coches sea superior al resto, los parámetros son potenciados o empeorados para garantizar un equilibrio entre los mismos.

Una vez está diseñado el balanceo, son los programadores los encargados de traducirlo a la lógica del programa. Se ocuparán de implementar las herramientas que necesitarán los diseñadores para equilibrar el juego. Este tipo de técnica aúna por tanto esfuerzo tanto de programadores como diseñadores para que el juego tenga un buen acabado.

2.4.- Generación procedimental de contenido

A la hora de desarrollar el contenido de un juego, éste puede ser elaborado previamente parte por parte, o creado en tiempo de juego a partir de recursos del programa. Este último caso, es conocido como generación procedimental de contenido.

Erróneamente llamado generación procedural de contenido (por su mala traducción del inglés), la generación procedimental de contenido[12] es el proceso por el cual un programa es capaz de, mediante sus propios algoritmos, elaborar contenido coherente necesario durante la ejecución del programa.

Una gran parte de los algoritmos de generación procedimentales son usados como generadores de contenido. Para ello, se implementan en su mayoría de forma recursiva buscando la mayor simplicidad posible, dado que a mayor complejidad del algoritmo, mayor probabilidad hay de que genere un comportamiento no determinista que dé lugar a incoherencias.

Su aplicación en el campo de los videojuegos es muy amplia. El contenido generado automáticamente sustituye los ficheros de datos que, sin generación procedimental, habrían sido creados durante la producción por el equipo de desarrollo. Se le ha dado uso por ejemplo, en la generación de mazmorras y calabozos para juegos de temática de rol y plataformas.

El primer juego en utilizar generación procedimental, llamado Beneath Apple Manor (Figura 3), es de 1978 y es considerado el primero perteneciente al género *roguelike* [4]. La meta del usuario es conseguir una manzana que se encuentra al final de la mazmorra. El contenido de la mazmorra, estaba generado mediante esta técnica.

El género *roguelike* se caracteriza por utilizar mucho este tipo de generación de mapas. Son juegos de exploración de mazmorras para un jugador donde los mapas, enemigos, y otros elementos son generados procedimentalmente.

[12] https://es.wikipedia.org/wiki/Generaci%C3%B3n_por_procedimientos

Un uso curioso lo encontramos en los árboles de familia del juego Crusaders Kings II de Paradox [5]. Es un juego de estrategia y rol ambientado en la Edad Media y nos pone al mando de una casa nobiliaria de la época.

En el, los personajes de la familia de la dinastía del jugador estaban agrupados en árboles y cada uno podía tener una personalidad diferente según distintos parámetros enteros.

Estos enteros, enmascarados en rasgos del carácter del personaje como el carisma o la inteligencia, suponían una modificación que acababa afectando a sucesos durante la partida que comprendían desde distintas tomas de decisiones durante una guerra a llevar a cabo una política matrimonial distinta.

Figura 3: Imagen in-game del juego Beneath apple manor

Dentro de los ejemplos básicos que podemos encontrar hay videojuegos como Dwarf Fortress [6] (juego que combina género *roguelike* con la construcción de ciudades), que utilizan algoritmos para generar mapas.

En el campo científico, podemos destacar el trabajo de Lara Cabrera et al. [7] donde realizan un sistema para la creación de mapas balanceados de un juego del género de estrategia en tiempo real.

En dicho artículo, describen el uso de algoritmos genéticos para que la distribución que persiguen sea uniforme y no favorezca a uno de los jugadores implicados, aplicando para ello una distribución Gaussiana sobre números enteros.

La generación procedimental de contenido enfocado en mapas contrasta con el diseño de escenarios nivel a nivel. Esta labor es realizada por los diseñadores. Para ello es necesario mucho tiempo dado que cada aspecto que contenía el nivel debe estar ubicado con un sentido y para hacer el nivel entretenido.

Sin embargo, para producciones tanto grandes como pequeñas, donde no se puede pagar un diseñador, o no se dispone de tiempo, se recurre a la generación procedimental para generar mapas. Aumentando la carga de programación en el desarrollo, aligeramos la carga de diseño y reducimos costes.

2.5.- Equilibrado de juegos por generación procedimental

El campo en que nos enfocaremos en este trabajo es en cómo podemos utilizar la generación procedimental para equilibrar juegos de dos jugadores.

En este campo se ha realizado poco trabajo aun y el objetivo marcado es ver si se puede realizar un sistema de balanceo de videojuegos mediante generación procedimental, que pueda tomar decisiones en tiempo real para que sea competitivo para ambos jugadores.

De este modo, la idea principal será desarrollar una biblioteca que sirva como base para facilitar el trabajo de los desarrolladores que quieran hacer un videojuego de generación procedimental para dos jugadores.

2.6.- Herramientas utilizadas para el desarrollo del proyecto

Para llevar a cabo el desarrollo de este proyecto hemos escogido como motor gráfico Unity y para desarrollar el código el IDE Visual Studio. Para el control de versiones utilizaremos un repositorio alojado en BitBucket.

2.6.1.- Unity 3D

Unity 3D [8] es un motor gráfico multiplataforma con el que se han desarrollado juegos de diversos géneros y complejidad. Posee una interfaz amigable para el

usuario y su desarrollo se lleva a cabo en C# o UnityScript (aunque este último lenguaje está en desuso en el motor).

La base del editor (Figura 4) es una arquitectura basada en componentes. Los objetos están constituidos por componentes que les otorgarán una funcionalidad u otra.

La base de su arquitectura es la clase MonoBehaviour[13]. Es la clase padre de la que heredan todos los scripts utilizados en el motor. También tenemos que destacar GameObject, que es la clase base de todas las entidades de Unity. Además de esta clase un proyecto Unity nos permite incorporar scripts que no hereden de ella.

A partir de él, se realiza una incorporación de componentes al GameObject en función de lo que necesitemos. Los componentes pueden ser de distintos tipos según su finalidad, y entre ellos algunos de los más conocidos son:

- Collider y todas sus variantes (utilizado para añadir una malla de colisión al objeto).
- Rigidbody, que añade al GameObject física para que pueda ser afectado por el motor de físicas de Unity.
- Script, que no es más que un archivo de código que permite realizar consultas y modificaciones sobre el objeto, y otros que se relacionen con el.

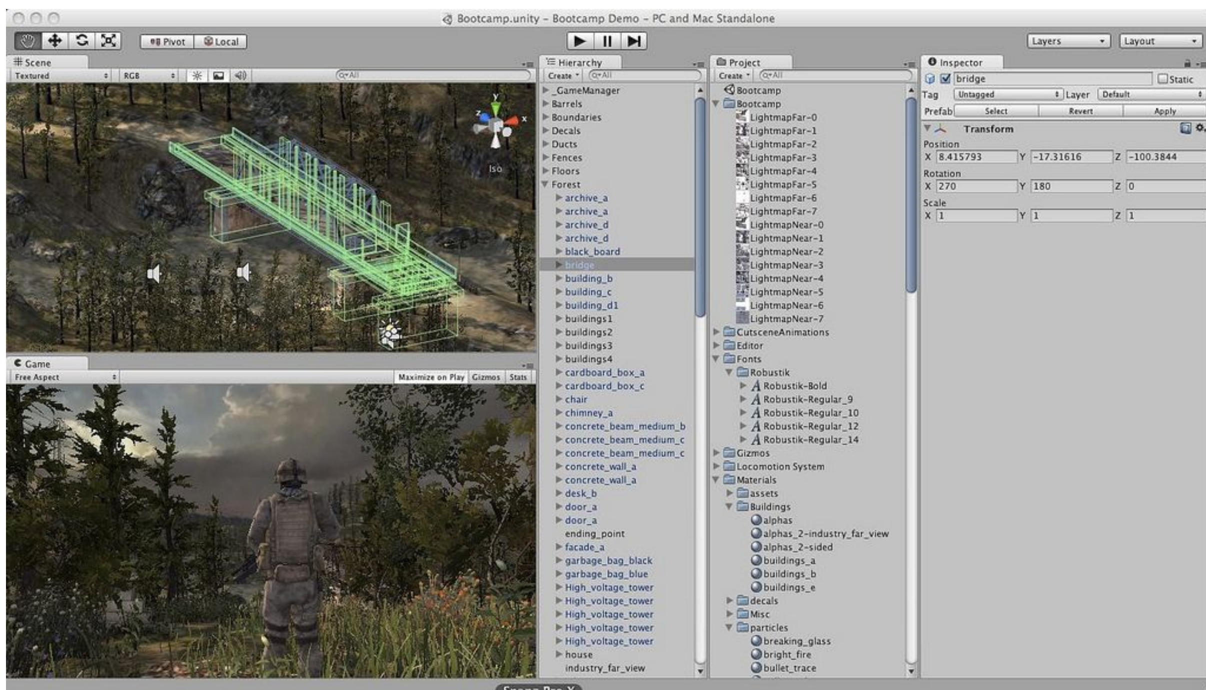

Figura 4: Editor de Unity 3D

[13] https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Además, su completa interfaz permite una personalización sencilla del motor de físicas, una gestión eficiente de los recursos lógicos y gráficos del proyecto, y la exportación del producto final a varias plataformas de entretenimiento además del ordenador, como puede ser Wii, PlayStation o Xbox.

2.6.2.- Visual Studio

Visual Studio es un entorno de desarrollo integrado (IDE) desarrollado por Microsoft y que ofrece soporte para desarrollar aplicaciones simples hasta complejos sistemas en una gran variedad de lenguajes, entre los que podemos citar C, C++ o C#.

Es precisamente en este último lenguaje, en el que desarrollaremos toda la lógica utilizada durante el proyecto. El motivo principal de escoger este IDE es que desde hace unos pocos años Unity y Microsoft llegaron a un acuerdo por el cual Microsoft daba soporte a la integración con Unity de su entorno de desarrollo, permitiendo con esto vincularlo al proyecto en desarrollo y poder hacer depuración del juego en tiempo real paralizando la ejecución del juego en tiempo real para poder depurar el código en busca de comportamientos extraños.

Esto, unido a la claridad y versatilidad de su entorno de depuración es lo que ha hecho que lo utilicemos durante el proyecto.

2.6.3.- BitBucket

BitBucket es un servidor de repositorios que permite llevar un control de versiones seguro y cómodo. Posee su propio cliente, además de la consola de comandos, y permite crear y gestionar tanto repositorios públicos como privados.

A la hora de elegir BitBucket, no ha habido motivos de peso respecto a otros clientes como GitHub, más que ya se tenían otros repositorios alojados aquí y se estaba familiarizado con el mismo.

CAPÍTULO 3.- IMPLEMENTACIÓN DE UN SISTEMA DE NIVELADO

Tras poner poner en contexto el proyecto, vamos a explicar qué problema queremos abordar. Una de las fases más complejas en el proceso de desarrollo de un videojuego es el equilibrado del mismo. Lo que buscamos es facilitar la implementación de este balanceo.

Lo primero que queremos hacer es definir cómo queremos que se haga. Por lo general las técnicas para obtener una igualdad entre los jugadores vienen predefinidas en el juego, el cual solo realiza ciertos cambios según detecte lo que le sucede a un jugador u otro.

Es decir, nos gustaría que el equilibrado del videojuego se produjese en tiempo real. Esto nos permitiría mitigar las desigualdades que surjan durante el juego entre los jugadores debido a que uno es más experto que otro.

Para abordar este problema, pensemos en un juego como el Tetris para dos jugadores (Figura 5). En este juego, cada uno de los usuarios va recibiendo piezas que tiene que ir alineando para aumentar su puntuación, eliminar bloques y así evitar que su pantalla se llene de piezas y pierda.

Figura 5: Imagen del tetris para dos jugadores de Wii

Cada vez que una pieza es colocada, otra nueva cae desde arriba con distinta forma. Y por supuesto, unas formas son más fáciles de encajar que otras.

El sistema que hemos implementado en este caso es ser capaz de, viendo que jugador se encuentra en peor condición de victoria, facilitarle su supervivencia, a la vez que intenta complicar la partida al jugador que gana.

Nuestro objetivo al implementar esta biblioteca es poder tener un sistema en el juego que permita jugar un jugador contra otro, a pesar de que el nivel de experiencia en este tipo de juegos entre ambos sea muy dispar.

Para conseguir esto, y haciendo alusión a lo explicado en el capítulo anterior, podemos hacerlo mediante generación procedimental de contenido, de tal modo que nos permita realizar un buen equilibrado. Esto es debido a que podríamos controlar el tipo de pieza que facilitamos a cada jugador mediante generación procedimental.

Una vez hemos visto con un ejemplo lo que queremos hacer, nos preguntamos: ¿Queremos que esto sea una implementación individual solo válida para un juego, o que pueda ser reutilizada para juegos que tengan ciertas características similares?.

La respuesta es sencilla. La idea que vamos a desarrollar es un conjunto de clases que actuarán como una biblioteca genérica, y que sirvan para cualquier juego que

tenga estas características comunes que hemos hablado. Es decir, apostamos por desarrollar una librería que pueda ser reutilizable en distintos juegos.

Esto permitirá simplificar el proceso de desarrollo de este equilibrado, y a su vez simplificar la tarea de la implementación del videojuego y sus costes.

3.1.- Qué es una biblioteca

Una biblioteca[14] es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.

Esta biblioteca, supone una capa intermedia entre el motor y los mapas diseñados para facilitar el trabajo del diseñador y del programador.

El desarrollo de la misma tiene como objetivo principal facilitar el nivelado de un juego de dos jugadores. Para ello debemos desarrollar un sistema que pueda llevar a cabo las funcionalidades que hemos visto previamente con ejemplos. Además, esta primera versión puede servir como base para ser ampliada en un futuro.

Una vez tengamos la biblioteca desarrollada, comprobaremos si se puede realizar un equilibrado efectivo del juego en tiempo real mediante generación procedimental de contenido.

3.2.- Estructura de la biblioteca

Para implementar la biblioteca, hemos buscado conseguir unas clases genéricas que manejen el grueso de la lógica que se encarga de tomar las decisiones relativas al equilibrado en el juego.

De este modo, la idea que tenemos es que sea una capa intermedia. De tal modo, la biblioteca podrá ser usada para reducir los costes de programación, pero nunca modificada.

[14] https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)

Esta capa intermedia se encargará de gestionar por dentro una lógica de toma de decisiones la cual luego el programador solo tenga que decir que cambios se tienen que realizar en el juego. Mediante esto, la biblioteca trabaja de forma aislada.

Así pues, podremos vincularla al juego que desarrollemos para utilizarla de tal modo que nos facilite su implementación. Además, la biblioteca puede ser utilizada junto a otras para ofrecer una mayor funcionalidad.

Debido a esto, deberíamos plantearnos cuál sería la frontera entre nuestro juego y la biblioteca, y en qué puntos interactúan. En la figura 6 podemos ver una primera perspectiva, a grandes rasgos de lo que queremos.

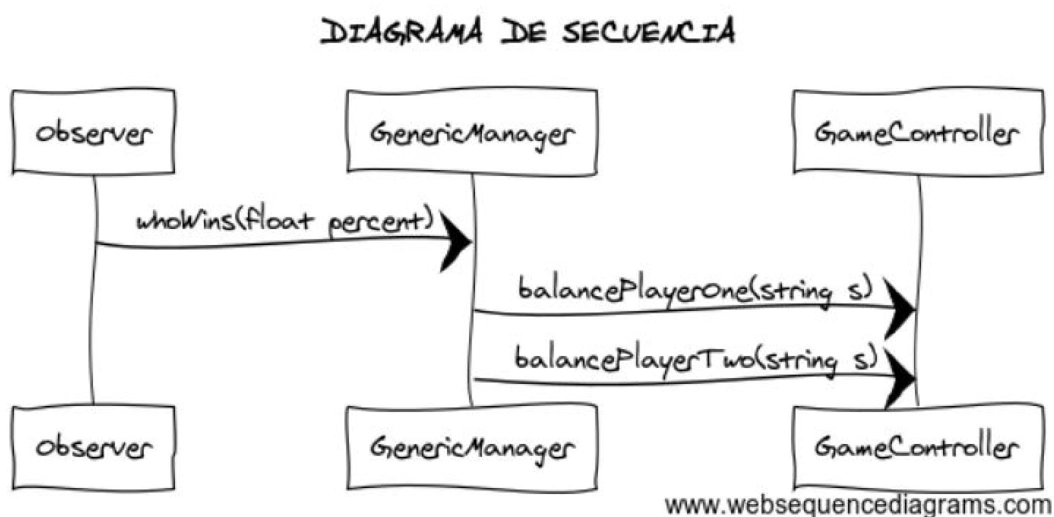

Figura 6: Diagrama básico de la biblioteca

Necesitaremos un manager como base de la arquitectura de toma de decisiones, así como una clase abstracta que permita luego al programador implementar los métodos de equilibrado.

Asimismo, también será necesario una clase que sondee los cambios que se producen en el nivel. Una vez detecte ciertos cambios, la idea es que los notifique al manager para que decida.

De este modo, podemos decir que nuestro modelo de clases quedaría distribuido como en la figura 7. Debemos destacar que esta arquitectura es común para cualquier motor de videojuegos en el que quiera ser implementada la biblioteca.

Esto hace que la labor del programador se reduzca solo a implementar cómo quiere equilibrar en concreto a uno de los dos jugadores. Por ejemplo, si el controlador utiliza la función de equilibrar al jugador uno, dentro de la función lo que puede tener por ejemplo, es reducir el tiempo que tarda en aparecer un obstáculo en el camino del jugador uno.

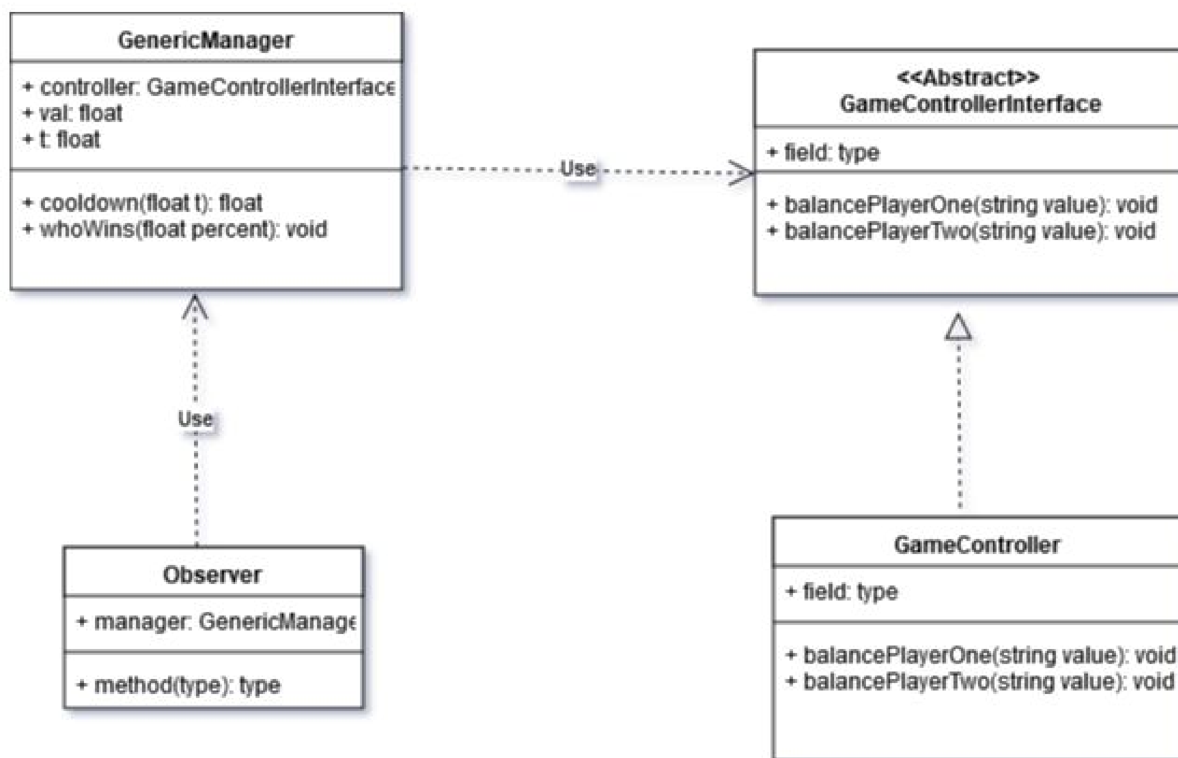

Figura 7: Diagrama de clases de la biblioteca

Además, dentro de la biblioteca también podemos encontrar la clase `GameControllerInterface`. Es una clase abstracta que incluye dos métodos para equilibrar la dificultad del jugador uno y del jugador dos.

La clase principal es `GenericManager`. Esta, se encargará de tomar las decisiones básicas respecto al juego en base a los parámetros que recibe.

Estos parámetros, serán suministrados por una clase que debe implementar el programador, llamada `Observer`. En ella, deberá incluir los elementos que le

resulten necesarios para hacer el equilibrado, y llamar al `GenericManager` para comunicarle este cambio.

Para que esta comunicación sea siempre la misma, el `GenericManager` codifica toda esta información mediante floats comprendidos entre -100 y 100, en función del valor que se le suministre decidirá que medidas tomar.

Además se encarga de controlar los delays de aplicación de los cambios para asegurar que no es una situación puntual y de unos pocos segundos. Esto queda codificado en base a un valor float n, que será el número de segundos en aplicarse.

Una vez el `GenericManager` toma una decisión, este se lo notifica a otra clase específica del juego implementada por el programador y que se llama `GameController`.
Esta clase hereda directamente de `GameControllerInterface`, que es abstracta.

De este modo, en el momento que el `GenericManager` vaya a realizar un cambio, llamará a una clase `GameControllerInterface`, y por extensión, `GameController`, y utilizará sus métodos para equilibrar el juego en base a los datos suministrados por la clase `Observer`.

El motivo por el cual las clases de `GameController` y de `Observer` no se encuentran implementadas es porque cada juego va a tener siempre unos datos propios, que afectarán al desarrollo del juego, y que no pueden ser parametrizados por la biblioteca.

En el caso de `GameController`, si que hay una relación de herencia con la biblioteca debido a que son las funciones “balancePlayerOne” y “balancePlayerTwo” que el programador implementará en `GameController` y de las que nuestro `GenericManager` hará uso.

Por otro lado, `Observer` solo tendrá que incorporar una referencia al `GenericManager` para llamar a sus funciones que modifican el valor del float que contabiliza en grado de victoria de un usuario sobre otro.

El programador tendrá que encargarse entonces de desarrollar un par de clases con los que alimentar a la biblioteca:

- ❏ Un observador que mida los datos y eventos relevantes en el videojuego, y en función de esos cambios se los pase a la biblioteca para que tome una decisión.
- ❏ Un actuador que heredará de una de las clases de la biblioteca. Esta herencia le aportará la capacidad de implementar dos métodos abstractos para realizar el equilibrado del juego.

Es mediante esta arquitectura, como realizamos el equilibrado que veremos en los juegos que se muestran en el siguiente capítulo.

Para ello, en base a si el float es negativo o positivo, el manager interpreta que jugador de los dos va perdiendo, y después da un margen de tiempo para comprobar que no es una situación casual.

3.3.- Implementación de la biblioteca en Unity 3D

Una vez hemos descrito la estructura que esperamos de nuestra biblioteca, llega el momento de introducirla en Unity 3D, que fue el motor escogido para el desarrollo del proyecto.

Para ello, el desarrollo tiene que seguir unas trazas genéricas. Esto es debido a que lo fundamental es que pueda ser reutilizable en distintos proyectos, como veremos en uno de los próximos capítulos, donde hablaremos de dos juegos desarrollados con la idea de probar el correcto funcionamiento de la biblioteca, así como su utilidad.

Para que la biblioteca sea implementada, utilizamos un prefab de Unity. Un prefab es un GameObject el cual tiene unos componentes básicos por defecto. Pueden ser creados por el desarrollador cuando sabemos que es un objeto que se va a reutilizar mucho. De este modo, nos evitamos montarlo cada vez que nos haga falta.

Una vez implementada la biblioteca, veremos que la labor tanto del programador como del diseñador a la hora de realizar el equilibrado se verá reducida en gran medida.

En Unity, la programación del juego se realiza mediante Scripts. Los Scripts se añaden como componentes a los objetos y les permiten tener un comportamiento en base a la lógica que se les ha programado.

Por lo tanto, tendremos cuatro Scripts, uno por cada clase del diagrama de la figura 7. `GenericManager`, `Observer` y `GameControllerInterface` heredarán de MonoBehaviour.

`GenericManager` utilizará dos funciones principales:

- ❏ `Cooldown(float n)`: es una función que facilitará un tiempo de espera pasado como parámetro.
- ❏ `whoWins(float percent)`: se encarga de, en función del valor float que recibe, y que como dijimos antes define que jugador va ganando:
    - ❏ -100 gana por mucho el primero.
    - ❏ 100 gana por mucho el segundo.
    - ❏ 0 implica equilibrio en el juego

El motivo de esta herencia es poder manejar todas las funcionalidades que nos ofrece el motor de Unity, y así poder enlazar los Scripts al prefab y poder configurarlo desde el editor..

Por último, el Script `GameController` deberá heredar de la clase `GameControllerinterface`, dado que lo que nos interesa es que el programador pueda implementar los métodos abstractos que se encuentran en esta clase.

Una vez implementada la biblioteca, la tarea del diseñador será la de:

- ❏ Añadir al nivel el prefab de la biblioteca que será utilizado.
- ❏ Introducir y modificar los parámetros del prefab, para obtener el comportamiento adecuado en el sistema, mediante ensayo-error.
- ❏ Comunicar al programador comportamientos no deseados y raros en el sistema para que vea si es debido a bugs de programación.
- ❏ Suministrar a los testers las versiones creadas para que las prueben y busquen fallos en el mismo

Por último, para conseguir esta genericidad, usaremos la clase base de Unity, de la cual heredan todos los Scripts: MonoBehaviour.

Esta clase padre, nos ofrece el comportamiento básico para utilizar todas las funcionalidades del motor. Nuestras clases heredarán de ella.

Así pues, el prefab tendrá el aspecto de la figura 8:

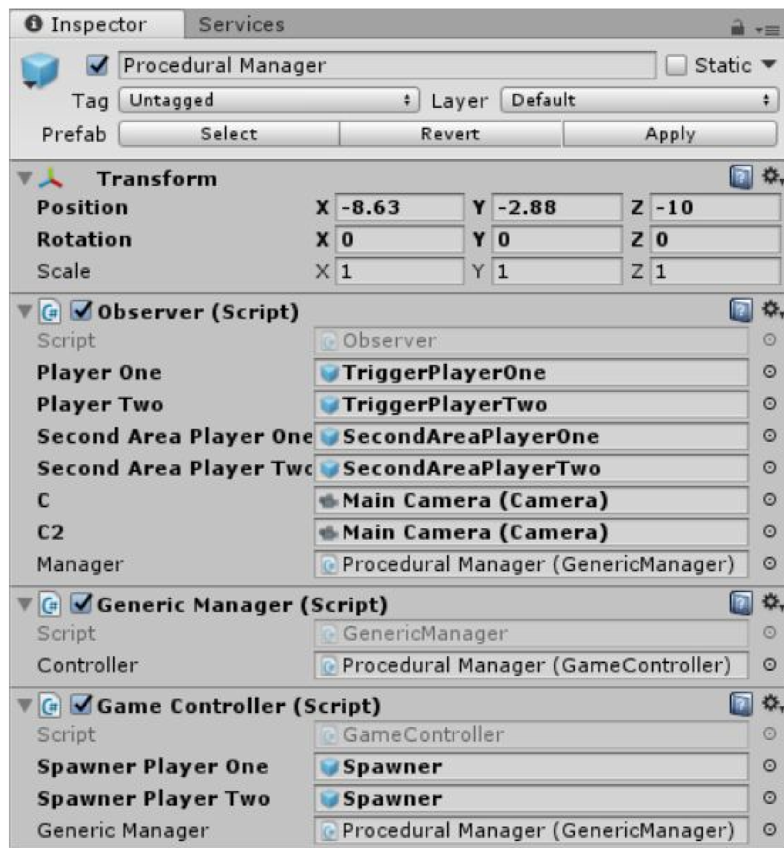

Figura 8: Imagen del prefab en un juego

Los campos que incluye tanto `GameController` como `Observer` son añadidos para la implementación de un runner procedimental.

CAPÍTULO 4.- JUEGOS DE PRUEBA

Si bien durante una gran parte del desarrollo del proyecto, nos hemos centrado en refinar el funcionamiento de la biblioteca para hacer a esta lo más genérica posible, debemos poner a prueba su versatilidad ante distintos tipos de juegos.

A pesar de que en la memoria hemos hablado primero de la biblioteca, el orden de desarrollo no fue ese. En primer lugar se desarrollaron dos juegos por generación procedimental y que precisaban de equilibrado.

En primer lugar, se desarrolló un runner por generación procedimental. A partir de este, se extrajeron las clases que consideramos comunes a juegos de este estilo para así sacar factor común diseñar la biblioteca anteriormente mencionada.

Posteriormente se desarrolló un juego arcade donde introdujimos esta biblioteca para ver su facilidad de integración en un proyecto nuevo.

Además, a la hora de desarrollar la biblioteca, se desarrolló un manager y clases específicas que gestionasen el equilibrado en el runner, para después ir poco a poco haciéndolas más genéricas.

De este modo, íbamos abstrayendo la lógica de nuestra biblioteca para que pudiese ser introducida en otro juego sin problema y a su vez resultase útil para el desarrollo y fácil de aplicar.

Para confirmar su correcto funcionamiento y facilidad de integración en nuevos proyectos, se han desarrollado dos juegos distintos que veremos a continuación.

4.1.- Runner procedimental

El primer juego en el que integramos la biblioteca pertenece al género de los *runner.* Las características de este estilo de juegos es que en sus niveles se avanza horizontalmente sorteando obstáculos sin poder detenerse.

Una variante de los *runner* consta de una profunda carga de diseño de niveles, prefijados, los cuales el jugador tiene que superar.

Dentro de este tipo encontramos Geometry Dash de RobTop Games [9], caracterizado por una estética minimalista y mecánicas simples que a medida que se avanza en el juego se van complicando (figura 9).

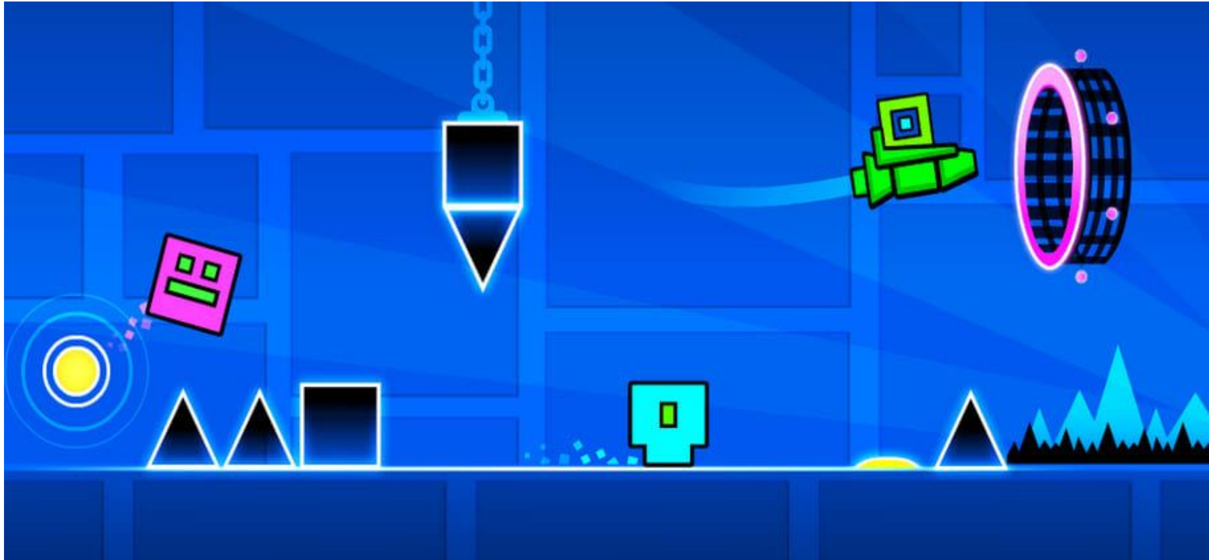

Figura 9: Imagen del juego Geometry Dash

Esto hace que el jugador desde niveles sencillos se vaya adaptando a la complejidad del juego a medida que avanza en él, pues busca provocar una sensación de superación en el individuo.

Por otro lado, dentro de los *runner* encontramos una variante donde los niveles en los que el jugador avanza son siempre distintos dado que utilizan generación procedimental de contenido.

Este tipo de juegos buscan simplemente que el jugador tenga que aguantar el mayor tiempo posible sobreviviendo en el nivel. A medida que el nivel avanza, el sistema lo va haciendo más complicado para poner a prueba las capacidades del jugador.

Ejemplos de este estilo de juegos son Jetpack Joyride de Halfbrik [10] el cual podemos ver en la figura 10, y que nos permite hacer una idea del tipo de juego del que estamos hablando.

Este juego es más completo en cuanto a lo que en videojuegos se conoce como coleccionables. Esto son pequeños logros como, por ejemplo, llegar a 1000 metros

recorridos. A base de acumular estos logros, el jugador va desbloqueando contenido.

Este sistema, es un modo de premiar al jugador tanto por su tiempo de juego como las destrezas adquiridas y resulta de utilidad para conseguir una mayor rejugabilidad del videojuego.

Figura 10: Parte de un nivel del juego JetPack Joyride

Como podemos ver, el personaje va avanzando, recogiendo monedas que le darán una puntuación, superando diversos tipos de obstáculos sin parar.

Para poner a prueba nuestro sistema, hemos desarrollado un juego similar, aunque más sencillo. Este consta de un personaje desplazándose a lo largo del escenario y donde tiene que ir esquivando obstáculos en el suelo y en el aire (figura 11).

Los jugadores se desplazan siempre hacia la derecha a una velocidad constante y viendo la situación del adversario por pantalla dividida. El mundo por el que corren los jugadores se va creando mediante *spawners* que son GameObjects que generan obstáculos conforme los jugadores avanzan. Estos obstáculos pueden estar tanto en el suelo como en el aire.

Cada jugador tiene su propio *spawner* que generará los obstáculos de su nivel. Estos, funcionan de modo independiente, y se les pueden modificar valores como el tiempo mínimo y máximo en que aparece un obstáculo, o cuantos obstáculos diferentes puede haber.

Cada vez que el jugador colisiona con un obstáculo, su avance se retrasa y se aproxima más al borde izquierdo de la pantalla. De este modo, si el jugador es alcanzado por el margen izquierdo de la pantalla, morirá.

De este modo, los dos usuarios comenzarán a jugar a la vez, y el ganador quedará determinado por aquel que consiga sobrevivir más tiempo sin morir.

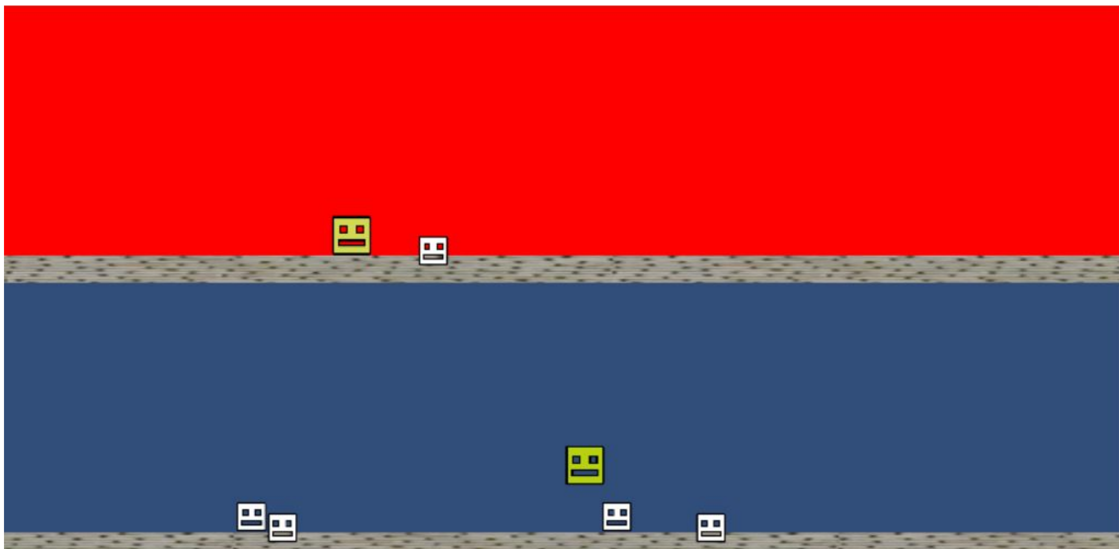

Figura 11: Imagen del runner (amarillo jugadores, blanco obstáculos)

Como lo que queremos es realizar un equilibrado en tiempo real para que en un juego de este estilo puedan competir jugadores tanto de niveles similares como dispares, hacemos uso de la misma introduciéndola en el sistema para que actúe como un balanceador.

En este primer caso, el observador sondea el estado de los jugadores en base a como de cerca están de ser eliminados. Cuanto más cerca esté el jugador del margen izquierdo de la pantalla, más cerca estará de ser eliminado.

Así pues, si el valor que recibe el manager es -100, querrá decir que el jugador 1 está a punto de perder, mientras que si es 100, será al contrario. Este valor se asume en base a unos triggers que detectan en que zona de la pantalla está el jugador.
De este modo, si el jugador 1 ha entrado en el trigger 2, que está en un punto medio entre el margen izquierdo y el centro, y el jugador 2 permanece en el centro sin

haber entrado aún en ningún trigger, el manager recibirá un -50, para que sepa que el jugador 1 se encuentra en clara desventaja.

De este modo, para ir detectando esto, el observador obtiene la posición en que se encuentra cada uno de los jugadores y en función de cual va peor, se lo hace saber al manager para que este tome las decisiones oportunas (complicar el nivel al otro jugador, facilitárselo al primero, etc)

Los niveles se complican del siguiente modo: el observador registra que un jugador va perdiendo y lo informa al manager. Este, notifica al controlador que necesita realizar cambios en la dificultad para complicar el nivel al que va en cabeza.

De este modo, el controlador accederá al generador de obstáculos del usuario que va en ventaja y aumentará el número de obstáculos que se generan en su nivel respecto al primero.

Esto lo hace mediante los parámetros de tiempo mínimo y máximo de spawn. Si el jugador va perdiendo, sus tiempos de spawn no se modificarán. Sin embargo, si el jugador lleva una clara ventaja sobre su oponente, el tiempo máximo y mínimo para que aparezca un obstáculo en el nivel disminuirán.

Por ejemplo, si un jugador se encuentra en la mitad de la pantalla, mientras que otro está a un tercio de distancia del lado izquierdo, el sistema hará que al jugador que va ganando le aparezcan más obstáculos.

Esto será debido a que el manager recibirá un valor -30 (supongamos que pierde el jugador 1) y el manager llamará al controlador para decirle que complique al nivel al jugador 2.

En ese momento, el controlador reducirá los tiempos de spawn del jugador 2 y esto provocará que en su nivel haya un mayor número de obstáculos que sortear.

Es así como conseguimos que una persona novata en el juego pueda competir con una que tiene experiencia, haciendo la partida interesante y entretenida.

4.2.- Juego arcade procedimental

El segundo juego que se desarrolló pertenece al género *arcade*. Este nombre proviene de las antiguas máquinas de los salones recreativos. Eran juegos pensados para que sus partidas tuviesen una duración corta y para poder continuar se tuviesen que comprar créditos introduciendo monedas.

Son juegos con un diseño sencillo, controles fáciles, corta y cuya dificultad va incrementándose a medida que se avanza en el mismo. Comparten similitudes, como buscar la puntuación más alta, fases de carga cortas, etc.

Sin embargo, este tipo de juegos no tienen un diseño y unas mecánicas similares como sucede en los *runner* si no que varía mucho de uno a otro.

Para hacernos una idea, podemos encontrar un juego clásico como Space Invaders de Taito Corporation [11] donde con nuestra nave debemos matar marcianos, mientras que en otro como Street Fighter de Capcom [12] lo que tendremos serán luchas entre dos personajes (figura 12).

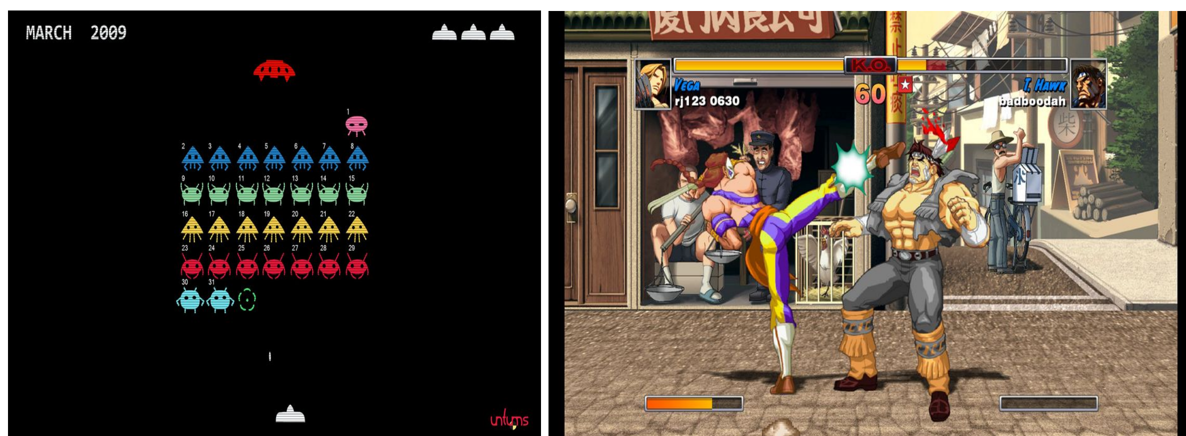

Figura 12: A la izquierda, Space Invaders. A la derecha, Street Fighter 2

En nuestro caso, el juego consta de un espacio dividido por una pared central, donde cada jugador se mueve de izquierda a derecha en un espacio sobre el que van cayendo objetos.

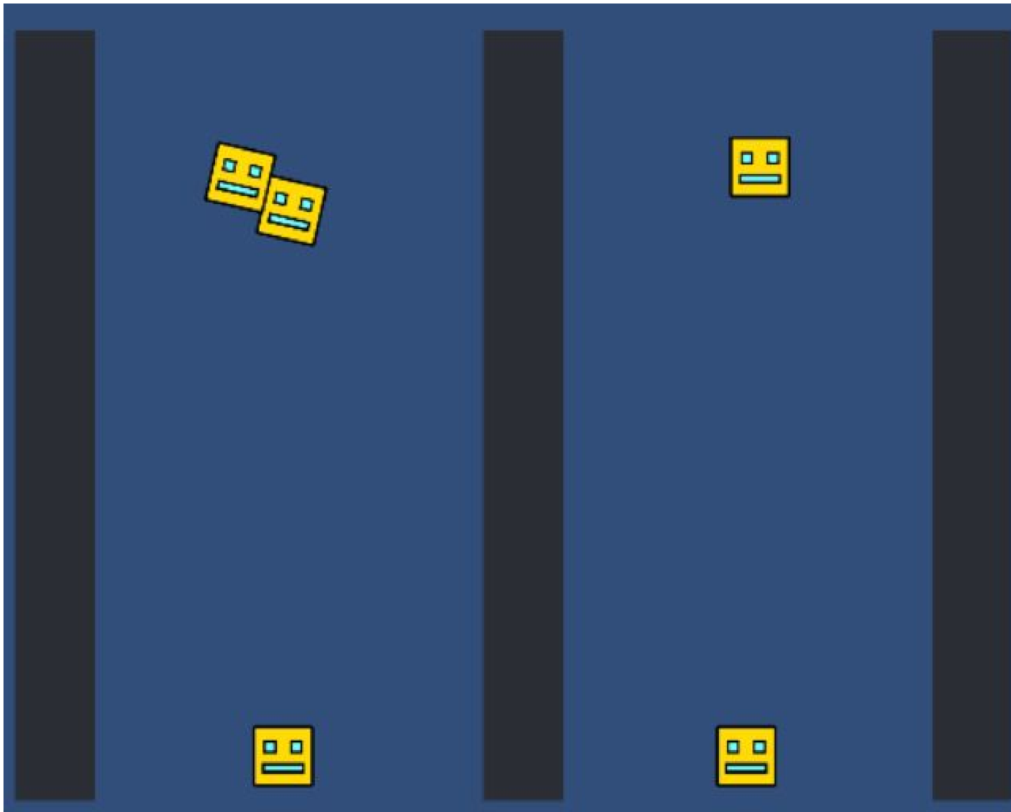

Figura 13: Imagen del juego (abajo jugadores, arriba obstáculos)

Los jugadores deben esquivar los objetos que caen para sobrevivir y tienen un máximo de golpes que pueden recibir (figura 13). Pierde el jugador que antes recibe ese número de impactos.

Los objetos son generados de un modo aleatorios por puntos de spawn distribuidos en la parte superior. Además, el número máximo de colisiones que puede tener un jugador antes de perder son 9.

Es decir, si nos encontramos en una situación de juego donde el jugador de la izquierda ha recibido 4 impactos y el de la derecha 0, el manager recibirá un valor -40 a través del `Observer`, de tal modo que se comunicará con el `GameController` para así poder decirle que disminuya el tiempo de spawn del jugador de la derecha, o que haga más largo el de la izquierda.

Esto propicia que sea otro ejemplo de juego de dos jugadores donde podemos incluir la biblioteca. En este caso, lo que sucede en el juego es que si un usuario ha recibido muchos impactos y el otro muy pocos o ninguno, el manager hará que el controlador aumente el número de objetos que debe esquivar el que va en cabeza.

Por otro lado, disminuirá los elementos a esquivar por parte del jugador que va peor posicionado para así facilitar que este pueda mantenerse en el juego y recuperarse frente a su contrincante.

Por ejemplo, si nuestro jugador de la izquierda ha recibido 5 impactos y el de la derecha 0, el sistema hará que al jugador de la derecha le caigan encima un mayor número de objetos para intentar equilibrar la situación.

4.3.- Evaluación empírica

El motivo de implementar este segundo juego era dilucidar la complejidad del uso de nuestra biblioteca en un proyecto nuevo.

Se ha podido determinar que dada la simplicidad de los juegos que en que se ha implementado la biblioteca,el rendimiento es despreciable. A la hora de evaluar este coste en juegos más complejos habría que realizar un estudio sobre los mismos en base a los parámetros a medir, tiempo de ejecución, etc.

Dado que en el primero de los dos juegos, la biblioteca se hizo generalizando las clases que gestionaban el nivelado, no se tenía una evidencia real de cómo de complejo podía ser.

Por ello, se incluyó la biblioteca en estos juegos para ver si resultaba sencillo utilizarla. Tras acabarlo, podemos concluir que se pudo implementar un sistema de balanceo de un modo sencillo.

Además, pudimos comprobar que queda perfectamente separada la biblioteca de las clases que la utilizan, dado que las clases de la biblioteca fueron solo usadas por otras.

Trás incluir la biblioteca en ambos juegos, pudimos ver que la implementación de las clases `Observer` y `GameController` se realiza de un modo bastante sencillo pues solo tenemos que tener en cuenta qué cambios queremos hacer en nuestro juego, y podemos olvidarnos de gestionar cuándo hacerlos.

Para ello, una vez tenemos los juegos implementados y funcionando, hemos procedido a realizar un cálculo del porcentaje de líneas que representan tanto la biblioteca como las clases implementadas.
Podemos sacar los siguientes datos:

- Del tamaño total del primer juego, el `Observer` y `GameController` que hacen uso de la biblioteca, es de un 10% en el primer juego (figura 14).
- Del tamaño total del primer juego, la clase `GenericManager` supone un 22% (figura 14).
- Del tamaño total del segundo juego, el `Observer` y `GameController` que hacen uso de la biblioteca, es de un 11% en el primer juego (figura 15).
- Del tamaño total del segundo juego, la clase `GenericManager` supone un 31% (figura 15).

Tras analizar estos datos, podemos concluir que el uso de la biblioteca supone un ahorro de código bastante grande respecto al tamaño del juego.

Figura 14: Porcentaje de código del runner

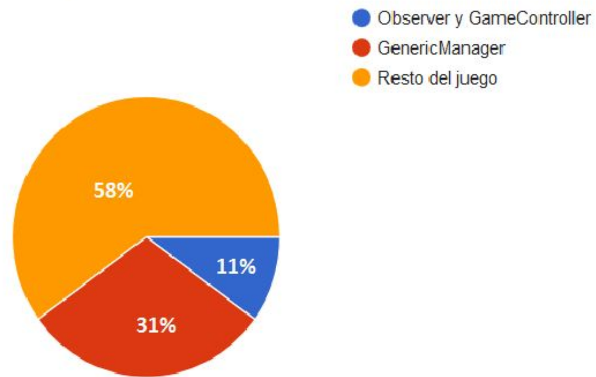

Figura 15: Porcentaje de código del arcade

CAPÍTULO 5.- EVALUACIÓN

Una vez hemos explicado como funciona la biblioteca que hemos implementado, y los juegos en los que la hemos usado, la siguiente fase que tenemos que realizar es la de evaluación.

5.1.- Realización de la evaluación

Nuestro objetivo será dar a probar los juegos expuestos en el capítulo anterior a distintas personas para ver si aprecian como el sistema complica al jugador que va con ventaja el nivel, y si por otro lado, el jugador que pierde nota que va recuperándose en el nivel.

Para ello se ha dado a probar uno de los juegos a 10 personas intentando juntar a un jugador que haya jugado a un juego parecido a los desarrollados (por ejemplo, runners) con otro que no los conozca, formulándoles las siguientes preguntas:

PREGUNTA	RESPUESTA
¿Juegas a videojuegos?	Si/No
En caso que si, ¿con qué frecuencia?	Ocasionalmente/Frecuentemente/ N/A
¿En el juego que has probado, has notado un cambio en la dificultad durante la partida?	He notado que se complicaba/ He notado que se hacía más fácil/ No he notado nada
¿Has notado que el juego ayudaba a que se equilibrase el nivel de los jugadores?	Si/No

Con estas preguntas, queremos por un lado ver el perfil de la gente que está jugando, y por otro, saber si el sistema de equilibrado es apreciable por los jugadores.

A todos los encuestados, se les sentaba delante de uno de los dos juegos, escogidos aleatoriamente. Después, jugaban una partida hasta que uno de los dos era eliminado.

A la hora de escoger las parejas, se procuró que se juntase a una persona familiarizada con alguno de estos dos tipos de juegos, frente a otra con menos experiencia en ellos.

Esto hará que haya más posibilidad de que la partida sea desigual y nuestro sistema de equilibrado en tiempo real tenga que actuar. De este modo puede ayudarnos a saber de un modo más seguro si funciona o no.

Tras jugar una partida con otra persona, se les pasaba un formulario para que contestasen a las preguntas anteriormente formuladas.

A las dos primeras preguntas, hemos obtenido los siguientes resultados (figuras 14 y 15):

¿Juegas a videojuegos?

10 respuestas

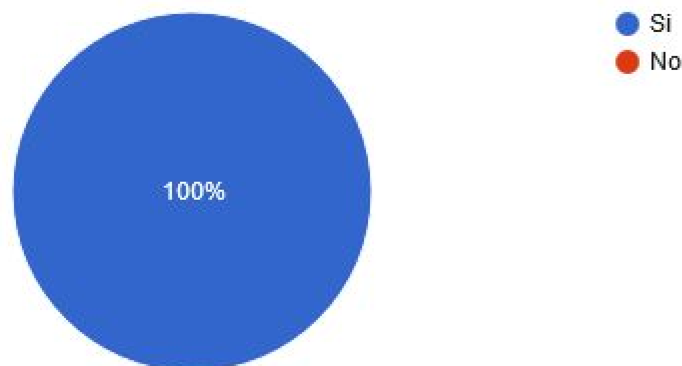

Figura 14

En caso que si, ¿con qué frecuencia?

10 respuestas

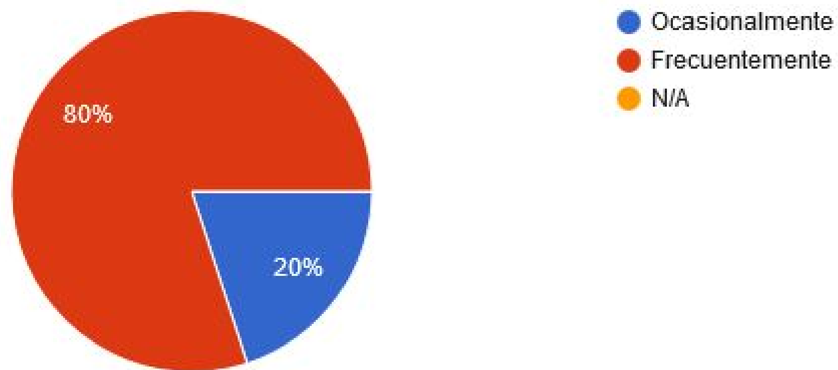

Figura 15

Para la tercera pregunta, el resultado queda más dividido (figura 16):

¿En el juego que has probado, has notado un cambio en la dificultad durante la partida?

10 respuestas

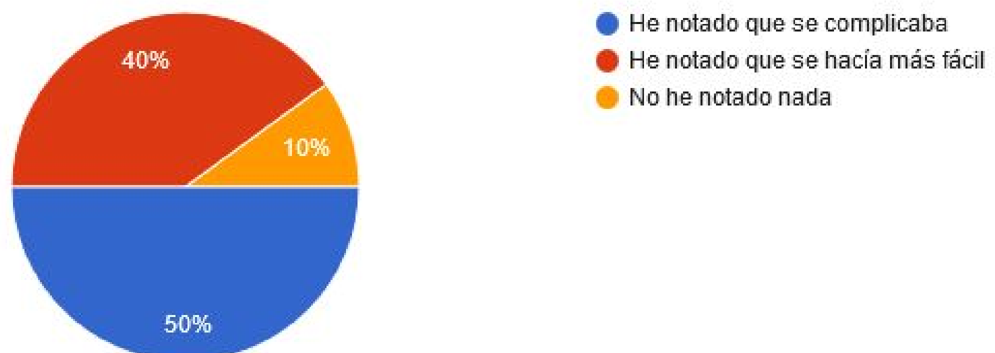

Figura 16

Podemos observar que un 90% de las personas notaron cambios en la dificultad del juego, frente a un 10% que sin embargo dijo no haber notado nada (perteneciente al grupo de los jugadores experimentados).

Esto puede ser debido a que tenía conocimientos sobre este tipo de juegos, y a que al estar acostumbrado a que los juegos de este estilo se compliquen conforme avanzan, no note la diferencia.
El 50% de los jugadores que notaban que el juego se complicaba coincidió con los que iban ganando durante la partida.
Del mismo modo, el 40% que apreciaron más facilidades fueron jugadores que estaban en desventaja y perdiendo.

Finalmente, para la última pregunta, los resultados obtenidos son los siguientes (figura 17):

¿Has notado que el juego ayudaba a que se equilibrase el nivel de los jugadores?
10 respuestas

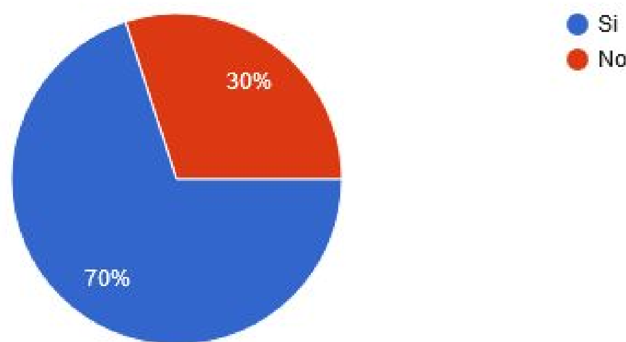

Figura 17

En esta última pregunta, el resultado está dividido entre un 30% que afirma no notar el equilibrado, frente a un 70% que si. Dentro de los que no lo notaron, un 20% eran jugadores experimentados y un 10% era un jugador novato, que no había tenido contacto con este tipo de juegos.

Dentro del 70% que lo notaron, un 40% fueron jugadores que iban perdiendo durante la partida. El 30% restante eran jugadores que tenían experiencia en estos juegos e iban ganando.

5.2.- Interpretación de resultados

Tras analizar las respuestas recopiladas durante las pruebas, podemos concluir lo siguiente:

- ❏ El perfil de la gente era de usuarios aficionados a los videojuegos, aunque en distintos grados.
- ❏ Una mayoría nota que el juego les favorece o dificulta su nivel.
- ❏ Un 70% de los encuestados perciben que el juego facilitaba el equilibrado.

Como conclusión principal, podemos sacar que el sistema de equilibrado del juego en tiempo real funciona y es apreciado por los jugadores.

Además, y como comentarios adicionales durante el transcurso de las pruebas, alguno de los jugadores que apreciaban que el juego se les complicaba, transmitía que no era justo para ellos respecto a la habilidad que tenía que se les complicase la victoria.

Algún otro jugador, por el contrario, comentó que hacía más interesante, e incluso “motivador” el hecho de aguantar la complicación.

5.3.- Trabajo futuro

Dentro de las posibles ampliaciones al presente proyecto que hemos desarrollado durante este tiempo podemos destacar las siguientes:

- ❏ Mejorar la toma de decisiones de la inteligencia artificial.
- ❏ Permitir mayor personalización por parte del diseñador a través del editor.
- ❏ Añadir un panel de configuración para que los diseñadores puedan trabajar de un modo más cómodo que con los prefabs.

CAPÍTULO 6.- CONCLUSIONES

Una vez terminado el desarrollo del presente proyecto, y después de analizar los datos que hemos obtenido a través de la fase de pruebas podemos sacar información que nos permita valorar el fin del trabajo.

Hemos obtenido una primera versión funcional de la biblioteca que puede ser utilizada por cualquier persona que quiera desarrollar un videojuego con las siguientes características:

- ❏ Generación procedimental de contenido.
- ❏ Juegos de dos jugadores.
- ❏ Género variado (runner, arcade, roguelike…)
- ❏ Desarrollo con Unity 3D

Esta biblioteca no se limita solo a este motor. El mismo esquema expuesto en el apartado 3.2. podemos extrapolarlo a otros motores como Unreal Engine 4, de tal modo que puedan ser utilizados en ellos.

La complejidad de la biblioteca radica en idear una arquitectura genérica que pueda cumplir los requisitos anteriormente expuestos sin problema.

Esto provocó que las clases que componían esta biblioteca se han ido reestructurando y refactorizando hasta tener la estructura actual, lo cual ha llevado a la desechar código que no era necesario, etc.

A pesar de ello, se ha obtenido una versión funcional de la biblioteca, y hemos podido comprobar su simplicidad para incluirla en el desarrollo de un videojuego como Unity.

Al tener el núcleo del equilibrado en ella, la labor de comunicarse con la biblioteca a través de las clases Observer y GameController se ve reducida a notificar cambios en la primera, y aplicarlos en la segunda. Esto se traduce en un menor número de líneas de código.

Otra conclusión que podemos destacar es que la elección del motor Unity 3D ha sido acertada, debido a que se tenía ya experiencia desarrollando con el mismo, y si se hubiese intentado realizar sobre un motor nuevo, la curva de aprendizaje habría demandado más tiempo del cual no se disponía.

Como conclusión final, y a través de los resultados obtenidos en la fase de pruebas, podemos concluir que el uso de tecnologías de generación procedimental para hacer un equilibrado en tiempo real pueden resultar útiles, a pesar de que algunas opiniones de los usuarios digan que no les resulta justo que se les iguale así a otros jugadores con menos nivel.

Una conclusión importante es poder ver que el uso de la generación procedimental ha podido aplicarse con éxito para este caso debido a que no habíamos encontrado referencias sobre el uso de la misma en el equilibrado.

Es por ello, que este trabajo puede suponer una pequeña toma de contacto para futuros trabajos, y que la biblioteca puede ser ampliada en un futuro de cara a ofrecer más funcionalidad.

CHAPTER 6.- CONCLUSIONS

Once the development of the present project is finished, and after analyzing the data that we have obtained through the testing phase, we can obtain information that allows us to value the end of the work.

We have obtained a first functional version of the library that can be used by anyone who wants to develop a video game with the following characteristics:

- ❏ Procedural generation of contents
- ❏ Two player games.
- ❏ Varied gender(runner, arcade, roguelike…)
- ❏ Development with Unity 3D

This library is not only limited to this engine. The same scheme showed in the section 3.2. can be extrapolated to other engines such as Unreal Engine 4, so that they can be used in them.

The complexity of the library lies in designing a generic architecture that can meet the abovementioned requirements without problems.

This caused that the classes that made up this library have been restructuring and refactoring up to have the current structure, which has led to discard the code that was not necessary, etc.

In spite of this, a functional version of the library has been obtained, and we have been able to verify its simplicity to include it in the development of a video game like Unity.

By having the core of the balancing in it, the task of communicating with the library through the Observer and GameController classes is reduced to notifying changes in the first, and applying them in the second. This results in fewer lines of code.

Another conclusion that we can highlight is that the choice of the Unity 3D engine has been successful, because we already had experience developing with it, and if we would had tried to perform it on a new engine, the learning curve would have required much more time, and it was not available.

As a final conclusion, and through the results obtained in the testing phase, we can conclude that the use of procedural generation technologies to do a real-time

balancing can be useful, although some users' opinions think that it is unfair that they are equal to other players with less level.

An important conclusion is to recognize that the use of procedural generation has been successfully applied for this case because we had not found references on its use in the balancing.

It is for this reason that this work can imply a small contact for future work. Furthermore, the library can be expanded in order to offer more functionality in future times.

REFERENCIAS

[1] Kent, S., “La gran historia de los videojuegos”, Penguin Random House Grupo Editorial España, 2016.

[2] Entertainment Software Association (ESA). "2017 Essential Facts About the Computer and Video Game Industry." The Entertainment Software Association, 2017. Disponible en http://www.theesa.com/wp-content/themes/esa/assets/EF2017_Design_FinalDigital.pdf.

[3] Chandler, H.M., “The Game Production Handbook 3rd Edition”, Jones and Bartlett Publishers, 2013

[4] Harrys, J., “@Play: Exploring Roguelike Games”, John Harris Editor, 2016

[5] Paradox Interactive, 2012

[6] Bay 12 Games, 2006

[7] Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A., “A Procedural Balanced Map Generator with Self-adaptive Complexity for the Real-Time Strategy Game Planet Wars”

[8] Menard, M. “Game development with Unity”, Delmar Cengage Learning, 2011

[9] Robtop Games, 2013

[10] Halfbrick, 2011

[11] Taito Corporation, 1978

[12] Capcom, 1993